Article
Stefan Wittmann · Jun 16, 2016 4m read

Producing JSON from SQL

Recently I have been posting some updates to our JSON capabilities and I am very glad that so many of you provided feedback. Today I would like to focus on another facet: Producing JSON with a SQL query.

Clearly, SQL is a significant concept to retrieve records from your relational model. Assume you want to query data and expose it as a simple JSON structure to a REST service. Usually, you have to query your data, then iterate over the result set and finally construct a JSON structure for each record. You have to write custom code.

We have added standard SQL functions to make it easier for you to directly produce JSON from a SQL query without writing code: JSONOBJECT and JSONARRAY . These two functions are new in Caché 2016.2.

Assume the following table Baking.Pastry for the examples:

Row	Type	Description
1	Choux Pastry	A light pastry that is often filled with cream
2	Puff Pastry	Many layers that cause the pastry to expand or puff when baked
3	Filo Pastry	Multiple layers of a paper-thin dough wrapped around a filling

JSONOBJECT

JSONOBJECT is a function that takes multiple key-value pairs and produces a JSON object.

```
SELECT JSON_OBJECT('pastry_type' : Type, 'pastry_description' : Description) AS pastr
yJSON FROM Baking.Pastry

Row    pastryJSON

----   ---------------

1      {"pastry_type" : "Choux Pastry", "pastry_description" : "A light pastry that i
s often filled with cream"}

2      {"pastry_type" : "Puff Pastry", "pastry_description" : "Many layers that cause
 the pastry to expand or puff when baked"}

3      {"pastry_type" : "Filo Pastry", "pastry_description" : " Multiple layers of a
paper-thin dough wrapped around a filling"}
```

In this example, the JSONOBJECT function produces one JSON object for each record. Each object contains two properties pastrytype and pastrydescription , as we provided two arguments to the function. Each argument consists of two parts, delimited by a colon:

1. The name of the key that should be injected into the object

2. The value associated with that key

This example sets static keys as I just provided a string literal, e.g. 'pastrytype' . For the value I am referring to a column, e.g. Type and whatever the content of that column is will be set as the value of the associated key. This is a common use case for constructing a JSON object, but by passing in a column for the key you can also create keys dynamically.

JSONARRAY

JSONARRAY works in a similar way. It constructs a JSON array and every argument that is passed in will push the corresponding value onto the array.

```
SELECT JSON_ARRAY(Type, Description) AS pastryJSON FROM Baking.Pastry


Row  pastryJSON

---- ---------------

1    ["Choux Pastry" , "A light pastry that is often filled with cream"]

2    ["Puff Pastry" , "Many layers that cause the pastry to expand or puff when baked
"]

3    ["Filo Pastry" , "Multiple layers of a paper-
thin dough wrapped around a filling"]
```

JSONARRAY is a pretty straightforward function. This example creates an array, which holds two elements for each row. The first item is populated by the value of the column Type, while the second item is filled with the value of the column Description.

Advanced scenarios

Maybe you have the requirement to create a more complex JSON structure. A value argument can be a nested JSONARRAY or JSONOBJECT function call, allowing you to construct nested JSON structures. Let ' s take the first example and wrap the JSON object in a header structure:

```
SELECT JSON_OBJECT('food_type' : 'pastry', 'details' : JSON_OBJECT('type' : Type, 'de
scription' : Description)) AS pastryJSON FROM Baking.Pastry


Row  pastryJSON

---- ---------------

1    {"food_type" : "pastry", "details" : {"type" : "Choux Pastry", "description" : "
A light pastry that is often filled with cream"}}

2    {"food_type" : "pastry", "details" : {"type" : "Puff Pastry", "description" : "
Many layers that cause the pastry to expand or puff when baked"}}

3    {"food_type" : "pastry", "details" : {"type" : "Filo Pastry", "description" : "
Multiple layers of a paper-thin dough wrapped around a filling"}}
```

There are more JSON SQL functions we plan to implement in future releases, but these two are a solid start. The

major use case is to construct simple JSON elements from your relational data without writing code. This way allows you to publish JSON from a system, even if you can ' t change the backend.

For creating more complex structures it is more efficient to build them with the new composition interface, which allows you to transform a persistent/registered object into a dynamic object/array. You can then easily modify the structure as you see fit and finally export it to a JSON string with a $toJSON() call. I will cover this topic in more detail in a future post.

#Best Practices #Caché #JSON #SQL